Package ‘d3Network’

October 13, 2022

Title Tools for creating D3 JavaScript network, tree, dendrogram, and Sankey graphs from R

Description This packages is intended to make it easy to create D3 JavaScript network, tree, dendrogram, and Sankey graphs from R using data frames. !!! NOTE: Active development has moved to the networkD3 package. !!!

Version 0.5.2.1

Date 2015-01-31

Author Christopher Gandrud

Maintainer Christopher Gandrud <`christopher.gandrud@gmail.com`>

URL `http://christophergandrud.github.io/d3Network/`

BugReports `https://github.com/christophergandrud/d3Network/issues`

License GPL (>= 3)

Depends R (>= 3.0.0)

Imports plyr, rjson, whisker

Suggests RCurl

NeedsCompilation no

Repository CRAN

Date/Publication 2015-01-31 17:29:42

R topics documented:

`d3ClusterDendro` *Create a D3 JavaScript Cluster Dendrogram graphs.*

Description

Create a D3 JavaScript Cluster Dendrogram graphs.

Usage

```
d3ClusterDendro(List, height = 2200, width = 900, heightCollapse = 0,
  widthCollapse = 0.5, fontsize = 10, linkColour = "#ccc",
  nodeColour = "#3182bd", textColour = "#3182bd", opacity = 0.8,
  diameter = 980, zoom = FALSE, parentElement = "body",
  standAlone = TRUE, file = NULL, iframe = FALSE,
  d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

`List`	a hierarchical list object with a root node and children.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`heightCollapse`	numeric proportion of the overall graph `height` to make the cluster dendrogram shorter by.
`widthCollapse`	numeric proportion of the overall graph `width` to make the cluster dendrogram narrower by.
`fontsize`	numeric font size in pixels for the node text labels.
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`nodeColour`	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
`textColour`	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`diameter`	numeric diameter for the network in pixels.
`zoom`	logical, whether or not to enable the ability to use the mouse scroll-wheel to zoom in and out of the graph.
`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is body.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.

`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file = NULL` then result is returned to the console.
`iframe`	logical. If `iframe = TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow framworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe = FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe = TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

Mike Bostock `http://bl.ocks.org/mbostock/4063570`.

Examples

```
## Not run:
# Download JSON data
library(RCurl)
URL <- "https://raw.github.com/christophergandrud/d3Network/master/JSONdata/flare.json"
Flare <- getURL(URL)

# Convert to list format
Flare <- rjson::fromJSON(Flare)

# Recreate Bostock example from http://bl.ocks.org/mbostock/4063570
d3ClusterDendro(List = Flare,
file = "FlareCluster.html", zoom = TRUE,
      fontsize = 10, opacity = 0.9,
      widthCollapse = 0.8)

## End(Not run)
```

`d3ForceNetwork` *Create a D3 JavaScript force directed network graph.*

Description

Create a D3 JavaScript force directed network graph.

Usage

```
d3ForceNetwork(Links, Nodes, Source, Target, Value = NULL, NodeID, Group,
  height = 600, width = 900, fontsize = 7, linkDistance = 50,
  linkWidth = "function(d) { return Math.sqrt(d.value); }", charge = -120,
  linkColour = "#666", opacity = 0.6, zoom = FALSE,
  parentElement = "body", standAlone = TRUE, file = NULL,
  iframe = FALSE, d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

`Links`	a data frame object with the links between the nodes. It should include the `Source` and `Target` for each link. These should be numbered starting from 0. An optional `Value` variable can be included to specify how close the nodes are to one another.
`Nodes`	a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the Source variable column in the `Links` data frame. Currently only a grouping variable is allowed.
`Source`	character string naming the network source variable in the `Links` data frame.
`Target`	character string naming the network target variable in the `Links` data frame.
`Value`	character string naming the variable in the `Links` data frame for how wide the links are.
`NodeID`	character string specifying the node IDs in the `Nodes` data frame.
`Group`	character string specifying the group of each node in the `Nodes` data frame.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`fontsize`	numeric font size in pixels for the node text labels.
`linkDistance`	numeric or character string. Either numberic fixed distance between the links in pixels (actually arbitrary relative to the diagram's size). Or a JavaScript function, possibly to weight by `Value`. For example: `linkDistance = "function(d){return d.value * 10}"`.
`linkWidth`	numeric or character string. Can be a numeric fixed width in pixels (arbitrary relative to the diagram's size). Or a JavaScript function, possibly to weight by `Value`. The default is `linkWidth = "function(d) { return Math.sqrt(d.value); }"`.
`charge`	numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`zoom`	logical, whether or not to enable the ability to use the mouse scroll-wheel to zoom in and out of the graph.

`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is body.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.
`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file = NULL` then result is returned to the console.
`iframe`	logical. If `iframe = TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow framworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe = FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe = TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for force directed networks `https://github.com/mbostock/d3/wiki/Force-Layout`.

Examples

```
#### Tabular data example.
# Load data
data(MisLinks)
data(MisNodes)

# Create graph
d3ForceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
               Target = "target", Value = "value", NodeID = "name",
               Group = "group", opacity = 0.4)

## Not run:
#### JSON Data Example
# Load data JSON formated data into two R data frames
library(RCurl)
MisJson <- getURL("http://bit.ly/1cc3anB")
MisLinks <- JSONtoDF(jsonStr = MisJson, array = "links")
MisNodes <- JSONtoDF(jsonStr = MisJson, array = "nodes")

# Create graph
d3ForceNetwork(Links = MisLinks, Nodes = MisNodes, Source = "source",
               Target = "target", Value = "value", NodeID = "name",
               Group = "group", opacity = 0.4)

## End(Not run)
```

d3Network *Function for creating simple D3 JavaScript force directed network graphs. DEPRICATED, USE d3SimpleNetwork*

Description

d3Network creates D3 JavaScript force directed network graphs. NOTE: it is depricated. It is currently a direct copy of d3SimpleNetwork

Usage

```
d3Network(Data, Source = NULL, Target = NULL, height = 600, width = 900,
  fontsize = 7, linkDistance = 50, charge = -200, linkColour = "#666",
  nodeColour = "#3182bd", nodeClickColour = "#E34A33",
  textColour = "#3182bd", opacity = 0.6, parentElement = "body",
  standAlone = TRUE, file = NULL, iframe = FALSE,
  d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

Data	a data frame object with three columns. The first two are the names of the linked units. The third records an edge value. (Currently the third column doesn't affect the graph.)
Source	character string naming the network source variable in the data frame. If Source = NULL then the first column of the data frame is treated as the source.
Target	character string naming the network target variable in the data frame. If Target = NULL then the second column of the data frame is treated as the target.
height	numeric height for the network graph's frame area in pixels.
width	numeric width for the network graph's frame area in pixels.
fontsize	numeric font size in pixels for the node text labels.
linkDistance	numeric distance between the links in pixels (actually arbitrary relative to the diagram's size).
charge	numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).
linkColour	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
nodeColour	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
nodeClickColour	character string specifying the colour you want the node circles to be when they are clicked. Also changes the colour of the text. Multiple formats supported (e.g. hexadecimal).
textColour	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).

`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is body.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.
`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file = NULL` then result is returned to the console.
`iframe`	logical. If `iframe = TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow frameworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe = FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe = TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for directed networks `https://github.com/mbostock/d3/wiki/Force-Layout`

`d3Sankey` *Create a D3 JavaScript Sankey diagram*

Description

Create a D3 JavaScript Sankey diagram

Usage

```
d3Sankey(Links, Nodes, Source, Target, Value = NULL, NodeID, height = 600,
  width = 900, fontsize = 7, nodeWidth = 15, nodePadding = 10,
  parentElement = "body", standAlone = TRUE, file = NULL,
  iframe = FALSE, d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

`Links`	a data frame object with the links between the nodes. It should have include the `Source` and `Target` for each link. An optional `Value` variable can be included to specify how close the nodes are to one another.

`Nodes`	a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the `Source` variable column in the `Links` data frame. Currently only grouping variable is allowed.
`Source`	character string naming the network source variable in the `Links` data frame.
`Target`	character string naming the network target variable in the `Links` data frame.
`Value`	character string naming the variable in the `Links` data frame for how far away the nodes are from one another.
`NodeID`	character string specifying the node IDs in the `Nodes` data frame.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`fontsize`	numeric font size in pixels for the node text labels.
`nodeWidth`	numeric width of each node.
`nodePadding`	numeric essentially influences the width height.
`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is body.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.
`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file = NULL` then result is returned to the console.
`iframe`	logical. If `iframe = TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow framworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe = FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe = TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for Sankey diagrams `http://bost.ocks.org/mike/sankey/`.

Examples

```
## Not run:
# Recreate Bostock Sankey diagram: http://bost.ocks.org/mike/sankey/
# Load energy projection data
library(RCurl)
URL <- "https://raw.githubusercontent.com/christophergandrud/d3Network/sankey/JSONdata/energy.json"
Energy <- getURL(URL, ssl.verifypeer = FALSE)
```

```
# Convert to data frame
EngLinks <- JSONtoDF(jsonStr = Energy, array = "links")
EngNodes <- JSONtoDF(jsonStr = Energy, array = "nodes")

# Plot
d3Sankey(Links = EngLinks, Nodes = EngNodes, Source = "source",
         Target = "target", Value = "value", NodeID = "name",
         fontsize = 12, nodeWidth = 30, file = "~/Desktop/TestSankey.html")

## End(Not run)
```

`d3SimpleNetwork` *Function for creating simple D3 JavaScript force directed network graphs.*

Description

`d3SimpleNetwork` creates simple D3 JavaScript force directed network graphs.

Usage

```
d3SimpleNetwork(Data, Source = NULL, Target = NULL, height = 600,
  width = 900, fontsize = 7, linkDistance = 50, charge = -200,
  linkColour = "#666", nodeColour = "#3182bd",
  nodeClickColour = "#E34A33", textColour = "#3182bd", opacity = 0.6,
  parentElement = "body", standAlone = TRUE, file = NULL,
  iframe = FALSE, d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

`Data`	a data frame object with three columns. The first two are the names of the linked units. The third records an edge value. (Currently the third column doesn't affect the graph.)
`Source`	character string naming the network source variable in the data frame. If `Source = NULL` then the first column of the data frame is treated as the source.
`Target`	character string naming the network target variable in the data frame. If `Target = NULL` then the second column of the data frame is treated as the target.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`fontsize`	numeric font size in pixels for the node text labels.
`linkDistance`	numeric distance between the links in pixels (actually arbitrary relative to the diagram's size).
`charge`	numeric value indicating either the strength of the node repulsion (negative value) or attraction (positive value).
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).

`nodeColour`	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
`nodeClickColour`	character string specifying the colour you want the node circles to be when they are clicked. Also changes the colour of the text. Multiple formats supported (e.g. hexadecimal).
`textColour`	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is body.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.
`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file` = `NULL` then result is returned to the console.
`iframe`	logical. If `iframe` = `TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow framworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe` = `FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe` = `TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

D3.js was created by Michael Bostock. See `http://d3js.org/` and, more specifically for directed networks `https://github.com/mbostock/d3/wiki/Force-Layout`

Examples

```
# Fake data
Source <- c("A", "A", "A", "A", "B", "B", "C", "C", "D")
Target <- c("B", "C", "D", "J", "E", "F", "G", "H", "I")
NetworkData <- data.frame(Source, Target)

# Create graph
d3SimpleNetwork(NetworkData, height = 300, width = 700,
                fontsize = 15)
```

d3Tree — *Creates a D3 JavaScript Reingold-Tilford Tree network graph.*

Description

Creates a D3 JavaScript Reingold-Tilford Tree network graph.

Usage

```
d3Tree(List, height = 600, width = 900, fontsize = 10,
  linkColour = "#ccc", nodeColour = "#3182bd", textColour = "#3182bd",
  opacity = 0.9, diameter = 980, zoom = FALSE, parentElement = "body",
  standAlone = TRUE, file = NULL, iframe = FALSE,
  d3Script = "http://d3js.org/d3.v3.min.js")
```

Arguments

`List`	a hierarchical list object with a root node and children.
`height`	numeric height for the network graph's frame area in pixels.
`width`	numeric width for the network graph's frame area in pixels.
`fontsize`	numeric font size in pixels for the node text labels.
`linkColour`	character string specifying the colour you want the link lines to be. Multiple formats supported (e.g. hexadecimal).
`nodeColour`	character string specifying the colour you want the node circles to be. Multiple formats supported (e.g. hexadecimal).
`textColour`	character string specifying the colour you want the text to be before they are clicked. Multiple formats supported (e.g. hexadecimal).
`opacity`	numeric value of the proportion opaque you would like the graph elements to be.
`diameter`	numeric diameter for the network in pixels.
`zoom`	logical, whether or not to enable the ability to use the mouse scroll-wheel to zoom in and out of the graph.
`parentElement`	character string specifying the parent element for the resulting svg network graph. This effectively allows the user to specify where on the html page the graph will be placed. By default the parent element is `body`.
`standAlone`	logical, whether or not to return a complete HTML document (with head and foot) or just the script.
`file`	a character string of the file name to save the resulting graph. If a file name is given a standalone webpage is created, i.e. with a header and footer. If `file = NULL` then result is returned to the console.

`iframe`	logical. If `iframe = TRUE` then the graph is saved to an external file in the working directory and an HTML `iframe` linking to the file is printed to the console. This is useful if you are using Slidify and many other HTML slideshow framworks and want to include the graph in the resulting page. If you set the knitr code chunk `results='asis'` then the graph will be rendered in the output. Usually, you can use `iframe = FALSE` if you are creating simple knitr Markdown or HTML pages. Note: you do not need to specify the file name if `iframe = TRUE`, however if you do, do not include the file path.
`d3Script`	a character string that allows you to specify the location of the d3.js script you would like to use. The default is `http://d3js.org/d3.v3.min.js`.

Source

Reingold. E. M., and Tilford, J. S. (1981). Tidier Drawings of Trees. IEEE Transactions on Software Engineering, SE-7(2), 223-228.

Mike Bostock: `http://bl.ocks.org/mbostock/4063550`.

Examples

```
## Create tree from R list
# Create hierarchical list
CanadaPC <- list(name = "Canada", children = list(list(name = "Newfoundland",
                    children = list(list(name = "St. John's"))),
               list(name = "PEI",
                    children = list(list(name = "Charlottetown"))),
               list(name = "Nova Scotia",
                    children = list(list(name = "Halifax"))),
               list(name = "New Brunswick",
                    children = list(list(name = "Fredericton"))),
               list(name = "Quebec",
                    children = list(list(name = "Montreal"),
                                    list(name = "Quebec City"))),
               list(name = "Ontario",
                    children = list(list(name = "Toronto"),
                                    list(name = "Ottawa"))),
               list(name = "Manitoba",
                    children = list(list(name = "Winnipeg"))),
               list(name = "Saskatchewan",
                    children = list(list(name = "Regina"))),
               list(name = "Nunavuet",
                    children = list(list(name = "Iqaluit"))),
               list(name = "NWT",
                    children = list(list(name = "Yellowknife"))),
               list(name = "Alberta",
                    children = list(list(name = "Edmonton"))),
               list(name = "British Columbia",
                    children = list(list(name = "Victoria"),
                                    list(name = "Vancouver"))),
               list(name = "Yukon",
                    children = list(list(name = "Whitehorse")))
))
```

```
# Create tree
d3Tree(List = CanadaPC, fontsize = 10, diameter = 500)

## Create tree from JSON formatted data
## dontrun
## Download JSON data
# library(RCurl)
# URL <- "https://raw.github.com/christophergandrud/d3Network/master/JSONdata/flare.json"
# Flare <- getURL(URL)

## Convert to list format
# Flare <- rjson::fromJSON(Flare)

## Recreate Bostock example from http://bl.ocks.org/mbostock/4063550
# d3Tree(List = Flare, file = "Flare.html",
#        fontsize = 10, opacity = 0.9, diameter = 1000)
```

`energy` *JSON data file of a projection of UK energy production and consumption in 2050.*

Description

JSON data file of a projection of UK energy production and consumption in 2050.

Format

A JSON file with two arrays nodes and `links`.

Source

See Mike Bostock `http://bost.ocks.org/mike/sankey/`.

`flare` *JSON data file of the Flare class hierarchy.*

Description

JSON data file of the Flare class hierarchy.

Format

A JSON file with two arrays name and `children`.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4063550`.

JSONtoDF *Read a link-node structured JSON file into R as two data frames.*

Description

`JSONtoDF` reads a JSON data file into R and converts part of it to a data frame.

Usage

```
JSONtoDF(jsonStr = NULL, file = NULL, array)
```

Arguments

`jsonStr`	a JSON object to convert. Note if `jsonStr` is specified, then `file` must be `NULL`.
`file`	character string of the JSON file name. Note if `file` is specified, then `jsonStr` must be `NULL`.
`array`	character string specifying the name of the JSON array to extract. (JSON arrays are delimited by square brackets).

Details

`JSONtoDF` is intended to load JSON files into R and convert them to data frames that can be used to create network graphs. The command converts the files into R lists and then extracts the JSON array the user would like to make into a data frame.

Source

Part of the idea for the command comes from mropa's comment on StackExchange: `http://stackoverflow.com/questions/4227223/r-list-to-data-frame`.

MisLinks *A data file of links from Knuth's Les Miserables characters data base.*

Description

A data file of links from Knuth's Les Miserables characters data base.

Format

A data set with 254 observations of 3 variables.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4062045`.

`MisNodes` *A data file of nodes from Knuth's Les Miserables characters data base.*

Description

A data file of nodes from Knuth's Les Miserables characters data base.

Format

A data set with 77 observations of 2 variables.

Source

See Mike Bostock `http://bl.ocks.org/mbostock/4062045`.

Index